A Two-Dimensional Topological Compactor With Octagonal Geometry

Paul de Dood† John Wawrzynek† Erwin Liu‡ Roberto Suaya‡

†Computer Science Division
University of California
Berkeley, CA 94720

‡Computer Science Laboratory
SRI International
Menlo Park, CA 94025

We present a two-dimensional layout compactor with octagonal geometry. We discuss layout optimization algorithms used within a topological framework. We present the results of comparing the compactor against a standard set of benchmarks. This version of the compactor uses an iterative greedy layout optimization algorithm with wire minimization and produces leafcells more compact than others published.

1 Introduction

Early work in compaction largely resulted in one-dimensional compactors [4,8,14]. These compactors, while fast, produce layout with area and wire length greater than would result from manual means. This is largely due to wires being treated as rigid in the direction perpendicular to the direction of compaction. Various methods of introducing wire jogs were attempted and produced better results [1]. However, the number of possible locations for jog insertion grows exponentially with the size of the cell.

Shin and Lo followed by Shin, Sangiovanni-Vincentelli, and Séquin attempted to overcome the shortcomings of one-dimensional compactors by employing quasi two-dimensional compaction strategies [11,12]. They used one-dimensional compactors as a preconditioner before attempting two-dimensional compaction of the cell. These approaches are a significant improvement over straight one-dimensional compactors.

Earlier work in two-dimensional compactors by Mosteller, Frey and Suaya [9] have shown the feasibility of their approach. Cells produced by their compactor were as good or better than hand-compacted cells. However, running times were very long.

The development of algorithms for single-layer routing from a topological description of the wires resulted in the development of our compactor. The *routability* of a design is based on a theorem proved by Maley [7] and independently by Valainis, et al. [13] and also Gao et al [2]. This *routability theorem* states:

> If the distance between every pair of bubbles of a cell is greater or equal to the topological design rule between them, then a design rule violation free routing of the wires exists.

As explained in section 2.1, bubbles are the primitives which define the features of the layout. The routability theorem allows us to discard the geometric positions of the wires while maintaining only their topological information. By enforcing the conditions of the routability theorem during layout optimization, we guarantee that a design rule violation free routing of the cell exists. After layout optimization, this routing is calculated and the new wire geometries are created. An advantage of this method is that during layout optimization, wires are continuously deformed (i.e. jogged) *in an optimal way*, without needing to calculate the exact jog points during layout optimization.

Our compaction process is divided into three phases:

1. Topological Extraction: Starting with an uncompacted symbolic layout of a cell in the OCT representation [3], a triangulation data structure is built storing the circuit's transistors and contacts along with a tiling of the cell area using triangles. The wiring topology is maintained but the wires' geometric positions are discarded. This data structure is used to represent the cell layout throughout the compaction process.

2. Layout Optimization: The layout is compacted through a sequence of movements of transistors and contacts. An incremental topological design-rule checker is used during layout optimization to ensure that sufficient space is reserved for later insertion of wires.

3. Wire Reconstruction: Once a compacted layout has been achieved, a topological wire router creates the geometric positions of the wires such that there are no design rule violations and that the length of each wire is minimum.

In section 2, we discuss the topological framework of the compactor and wire router. Section 3 discusses our optimization algorithms. In section 4, we compare the results of our compactor, named BOLT, against the results of other compactors using a standard set of benchmarks.

2 Topological Framework

2.1 Layout Primitives

A *bubble* B is composed of one or more octagons, such that all octagons have a common center, and each octagon has a radius and a layer associated with it. A *wire segment* is composed of one or more stretched octagons, such that all stretched octagons have a common center line, and each stretched octagon has a radius and a layer associated with it. A *wire* W connecting two *terminal* bubbles is a concatenation of wire segments such that all the wire segments have exactly the same layers and the corresponding radii, and for each layer on the wire, the terminal bubbles contain the same layer as the wire, with equal or larger radii. CMOS transistors are constructed from 10 bubbles and 5 wires, as shown in Figure 1.

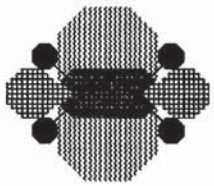

Figure 1: Bubble representation of a CMOS transistor.

2.2 The Triangulation Database

The layers of a cell layout are assigned to several *planes*. Each plane contains that set of layers that interact with each other. Each plane is then divided into a set of non-overlapping *triangles* [10]. This basic triangulation technique is adapted to store the topological layout information as explained in detail in [13]. The bubble centers are linked by the *edges* of the triangulation and each wire is recorded as two terminal bubbles and a list of triangle edges that the wire crosses.

An important operation in this data structure is *edge swapping*; the existing diagonal of a convex quadrilateral in the triangulation is replaced by the other diagonal. Another important operation is the *forcing* of an edge to link two bubbles in the layout. To force an edge, the *envelope* is first identified. The envelope is a sequence of triangles which enclose the line joining the two bubbles. The edges contained by the envelope are then swapped until there is an edge linking the two bubbles. It has been shown in [5] that such a sequence of swaps always exists. Once an edge is forced, it directly links two bubbles.

The *topological design rule* between two bubbles is the minimum separation of the bubbles such that the wires on the edge linking the bubbles can be linearly arranged without causing a spacing design rule violation. If the distance between the two bubbles is less than their topological design rule, there is a *topological design rule violation* (DRV) between the bubbles.

2.3 Routing

By applying the routability theorem, the layout optimization algorithm guarantees that there is sufficient room to route the wires. Wire routing, done after layout optimization, is the reconstruction of the wire geometries from the wire topologies stored in the triangulation database. Through a process of wire tightening and design rule violation removal, the router generates the optimum[1] (minimum length) route of each wire for the given topology. The details of the wire routing algorithm are discussed in [6].

2.4 Bubble Moving

The algorithm for moving a bubble in the layout is[2]:

```
for each plane p {
  move bubble b0 in plane p;
  seal hull around b0;
  check for DRV's;
  while ( DRV's ) do {
    move b0 back sufficiently to
    remove DRV;
    seal hull around b0;
    check for DRV's;
  }
}
```

The *hull* around a moving bubble B_0 is the set of bubbles that are sufficiently close to B_0 such that they *could* have a DRV with B_0. To determine the set of hull bubbles all triangles that intersect B_0, called *hull triangles*, must be *sealed*. As shown in Figure 2, the edge of the hull triangle opposite B_0 is the *hull edge* E_h.

A hull triangle $\triangle B_0 B_1 B_2$ is sealed if B_0 cannot have a DRV with any bubble in the sector $\angle B_1 B_0 B_2$. Although

[1] The use of non-curvilinear geometry results in multiple optimum routes each having the same wire length.

[2] For clarity, we make some simplifications in this discussion.

there are no bubbles within $\triangle B_0 B_1 B_2$, the region beyond the hull edge E_h could contain any arbitrary set of bubbles. The sealing criterion we use is:

$$DR(B_0, B_1) + |E_h| < |E_2|$$

or

$$DR(B_0, B_2) + |E_h| < |E_1|$$

$|E|$ is the distance between the two endpoints of the edge. $DR(B_0, B)$ is the topological design rule from B_0 to B, where each octagon of B is assumed to be an octagon of whichever layer results in the largest possible design rule between B_0 and B, instead of it's actual layer. If this criterion is satisfied, then no bubble in the sector $\angle B_1 B_0 B_2$ can have a DRV with B_0, and the triangle is said to be *sealed*. The proof of this statement is beyond the scope of this paper.

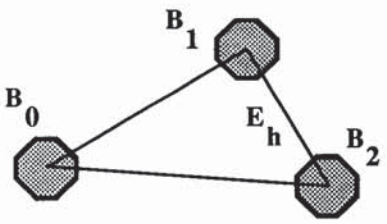

Figure 2: A hull triangle of the moving bubble B_0.

If a hull triangle is not sealed, the hull edge E_h is swapped. This results in the replacement of the hull triangle by two new hull triangles, as shown in Figure 3. This hull expansion terminates quickly because the number of hull triangles needed before B_0 seals is on average constant, regardless of the number of bubbles in the cell being compacted [13].

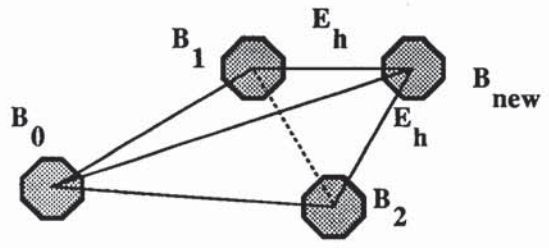

Figure 3: The result of swapping a hull edge.

Once the hull of B_0 is sealed, B_0 is checked for DRV's against every hull bubble. B_0 is said to *collide* with those bubbles which have a DRV to B_0. If B_0 collides with any bubbles, it is moved back sufficiently to remove the DRV(s).

A subset of bubbles, such as the set of bubbles that define a transistor, may be constrained to move as a unit. To perform this move, the bubbles are first ordered such that those bubbles furthest in the direction of motion are moved first, while those bubbles trailing behind the front bubbles are moved afterwards. The bubbles are then moved, in order, in a given direction as far as dictated by the layout optimization algorithm. If any bubble has a collision which prevents it from moving the full distance, the other bubbles are moved back until all bubbles in the group have moved by the same amount. Note, it is not necessary to check for DRV's during the backward move as design rule checking was already performed during the forward move.

3 Layout Optimization Algorithms

3.1 Flow Algorithm

The *flow* algorithm attempts to move all bubbles as far as possible in a given direction. The bubbles, ordered by position, are moved one at a time until they collide with another bubble or the border of the cell.

By using the flow algorithm in two perpendicular directions the result is similar to the result of one-dimensional compactors with the addition of automatic jog insertion. However, the ability of bubbles to move in any direction, regardless of the number of wires attached to them can present a problem. Where in non-topological compactors the wires are usually too rigid and need to have jogs introduced, in our topological compactor the automatic jog insertion of wires allows the bubbles to move too freely. The wires stretch to allow their terminating bubbles to move as far as possible in one direction and can interfere with compaction in the perpendicular direction. In order to offset this problem, we developed a wire length reduction algorithm.

3.2 Wire Length Reduction

To adequately reduce cell area, it is necessary to reduce total wire length. Reducing wire length is also useful in reducing the parasitic resistance and capacitance on critical wires.

The wire length reduction algorithm, called *relax*, also moves bubbles one at a time in a given direction. However, instead of moving a bubble B_0 as far as possible, B_0 is moved to its *preferred* point. The preferred point for B_0 is the point at which the length of the wires attached to B_0—weighted by their width and layer—is locally minimized.

Because the geometric positions of wires is not known until after routing, it is necessary to estimate the geometric positions of the wires in order to reduce the wires' length.

The list of edges that a wire topologically crosses is the *edge list* of that wire. Every edge which the wire

crosses topologically, the wire also crosses geometrically. The wire estimation routine attempts to determine this *crossing point* for one of the edges in the wires' edge list.

A wire cannot cross an edge at a point where the wire would have a DRV to either of the bubbles linked by the edge. The remaining portion of the edge, called a *window*, defines the interval at which the wire could cross the edge. Figure 4 shows a partial edge list of a wire with the windows highlighted on the edges. Starting from the moving bubble and stepping through the edges in the edge list, the window for each edge in the edge list is determined until a window is found which cannot be seen by the bubble through all of the previous windows. The last window which the bubble can see is the *wrapping window* and the edge which the wrapping window is on is the *wrapping edge*. At the wrapping window, the wire turns out of sight either to the left or to the right. If the wire turns to the right, as is the case in Figure 4, the rightmost point of the wrapping window is the estimate of the crossing point for the wrapping edge. This point is circled in the figure. If the wire turns left, the leftmost point of the window is the estimate of the crossing point for the wrapping edge. Once an estimate for the crossing point of an edge is determined, it is assumed that the wire will pull the moving bubble towards that point. The force of the pull is dependent on the layer of the wire, as well as the wire's width. Currently, the capacitance per unit length of the wire times the wire's width is used as an estimate of the magnitude of the force.

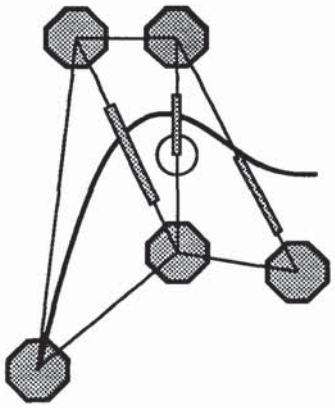

Figure 4: Estimating wire position

The sum of the forces exerted on a bubble by wires attached to the bubble will pull the bubble towards its preferred point.

3.3 Iterative Optimization Algorithm

The iterative optimization algorithm uses both the flow and relax algorithms. For a given direction, the bubbles are moved as far as possible in that direction according to the flow algorithm. Then, the bubbles are moved back in the opposite direction according to the relax algorithm. The net result is that the cell is as narrow as possible along the axis of compaction, *but those bubbles that did not contribute to the width in that direction (i.e. the bubbles not on the critical path) are moved such that wire length is minimized.* This process is repeated for different octagonal directions. The ordering of compaction directions is under user control, but we have found that the order which produces the densest layout is a spiral (i.e. E, NW, S, NE, W, SE, N, SW). Of course, the spiral pattern can be repeated as long as the cell area continues to decrease.

We have also been developing and experimenting with other optimization algorithms within our topological framework. These other algorithms include simulated annealing and other statistical methods, as well as rule-based approaches.

4 Results

We compared the results of the iterative optimization algorithm as described above with the results of other compactors using the benchmarks of the ICCD '87 compaction workshop [1] and the results of the new MACS compactor[11]. These benchmark examples include two leaf cells entitled *afa* and *afavg*, as well as some hierarchical multipliers.

The *afa* cell compacted by BOLT is shown in Figure 5. For clarity, the wells are not shown in the figure, though they are present.

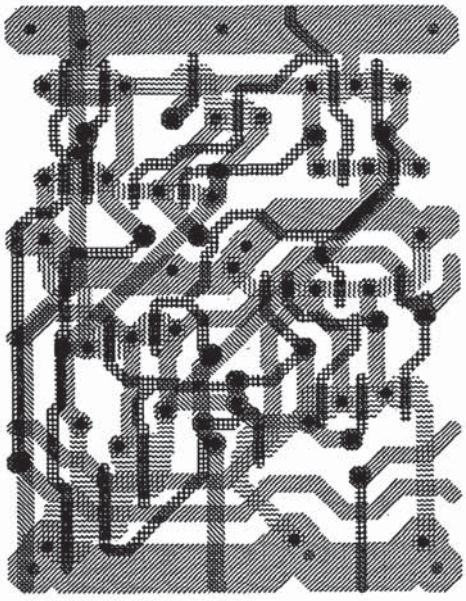

Figure 5: Compacted benchmark cell of an AND-full-adder (afa)

The area comparisons for the two leaf cells and the 2 by 2 multiplier are given in Table 1. The area total includes that area needed for wells.

Because the code has not yet been sufficiently optimized for speed, running times for compaction are significantly greater than other compactors. However, as we optimize the code, we expect to obtain running times comparable to other compactors, though still greater.

Compactor	**Area (μ^2)**
	afa
BOLT	123.45 x 162.9 = 20110.0
MACS.2D	130.5 x 169.5 = 22119.8
MACS	143 x 166 = 23738.0
Zorro	140.5 x 171 = 24025.5
SPARCS	157 x 180 = 28260.0
Symbolics	160 x 189 = 30240.0
	afavg
BOLT	114.9 x 156.75 = 18010.6
MACS.2D	129.75 x 144.75 = 18781.3
Zorro	128.5 x 151 = 19403.5
MACS	142 x 145 = 20590.0
SPARCS	157 x 151 = 23707.0
Symbolics	154 x 154 = 23716.0
	mul2x2
BOLT	264.6 x 233.7 = 61837.0
MACS	309 x 252 = 77868.0
Zorro	312 x 252 = 78624.0
SPARCS	343 x 255 = 87465.0
Symbolics	370 x 270 = 99900.0

Table 1: Area comparison of benchmark examples

5 Conclusion

In this paper, we present new compaction algorithms used within a topological framework. These algorithms have been implemented in the OCT environment in the form of an experimental leafcell compactor. The algorithms provide full two-dimensional compaction using octagonal geometry and automatically make optimal wire jogs. Preliminary results show that our iterative greedy algorithm with wire length reduction produces leafcells more compact than all others published. More advanced layout optimization algorithms are currently under development.

While this compactor may be used as a stand-alone leafcell compactor, our goal is to develop a compactor for hierarchical collections of cells. In hierarchical compaction the total area is often strongly dependent on the inter-cell connections. A difficult tradeoff must be faced between "stretching" cells to match with their neighbors and "river routing" between them. Because this compactor automatically jogs wires in the process of compacting the layout, this tradeoff is easily resolved.

References

[1] D. G. Boyer. Symbolic Layout Compaction Benchmarks - Results. In *IEEE International Conf. on Computer Design*, 1987.

[2] F. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rulling, and C. Storb. On Continuous Homotopic One Layer Routing. In *Proceedings of 4th Annual Symposium on Computational Geometry*, 1988.

[3] D. S. Harrison, P. Moore, R. L. Spickelmier, and A. R. Newton. Data Management and Graphics Editing in the Berkeley Design Environment. In *4th International Conference on Computer-Aided Design (ICCAD 86)*, pages 24–27, New York, New York, November 1986. Institute of Electrical and Electronic Engineers.

[4] Min-Yu Hsueh. *Symbolic Layout and Compaction of Integrated Circuits*. PhD thesis, University of California at Berkeley, Berkeley, California, December 1979.

[5] E. Liu. *Two Dimensional IC Layout Compaction*. PhD thesis, University of Calgary, Calgary, Alberta, Canada, December 1986.

[6] E. Liu, P. de Dood, R. Suaya, and J. Wawrzynek. A Topological Framework for Compaction and Routing. In *Advanced Research in VLSI*, 1991.

[7] F. M. Maley. *Single Layer Wire Routing*. PhD thesis, Massachusetts Institute of Technology, August 1987.

[8] R. C. Mosteller. REST—A Leaf Cell Design System. In *Very Large Scale Integration*, University of Edinburgh, Edinburgh, Scotland, August 1981. Academic Press.

[9] R. C. Mosteller, A. Frey, and R. Suaya. 2-D Compaction - a Monte Carlo Method. In *Proceedings of the Stanford Conference on Advanced Research in VLSI*. MIT Press, 1987.

[10] F. Preparata and M. I. Shomos. *Computational Geometry An Introduction*. Springer-Verlag, 1985.

[11] H. Shin and C. Lo. An efficient two-dimensional layout compaction algorithm. In *26th ACM/IEEE Design Automation Conference*, 1989.

[12] H. Shin, A. Sangiovanni-Vincentelli, and C. H. Sèquin. Zone-Refining Techniques for IC Layout Compaction. *IEEE Transactions on CAD of ICs and System*, 9(2), February 1990.

[13] J. Valainis, S. Kaptanoglu, E. Liu, and R. Suaya. Two Dimensional IC Layout Compaction Based on Topological Design Rule Checking. *IEEE Transactions on CAD of ICs and System*, 9(3), March 1990.

[14] N. Weste. Virtual grid symbolic layout. In *Proc. 18th Design Automation Conf.*, pages 225–233, 1981.